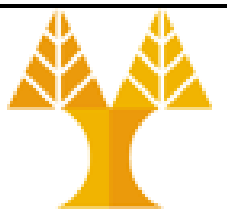

ΕΡΓΑΣΤΗΡΙΟ #7

ΜΕΡΟΣ Α΄: OVERVIEW OF TRANSACTION MANAGEMENT

1. (Exercise 16.1) Give brief answers to the following questions:

a. What is a transaction? In what ways is it different from an ordinary program (in a language such as C)?
b. Define these terms: atomicity, consistency, isolation, durability, schedule, blind write, dirty read, unrepeatable read, serializable schedule, recoverable schedule.

Answer:

a. A transaction is an execution of a user program, and is seen by the DBMS as a series or list of actions. The actions that can be executed by a transaction include reads and writes of database objects, whereas actions in an ordinary program could involve user input, access to network devices, user interface drawing, etc.
b. **Atomicity** means a transaction executes when all actions of the transaction are completed fully, or none are. This means there are no partial transactions (such as when half the actions complete and the other half do not).

Consistency involves beginning a transaction with a 'consistent' database, and finishing with a 'consistent' database. For example, in a bank database, money should never be "created" or "deleted" without an appropriate deposit or withdrawal. Every transaction should see a consistent database.

Isolation ensures that a transaction can run independently, without considering any side effects that other concurrently running transactions might have. When a database interleaves transaction actions for performance reasons, the database protects each transaction from the effects of other transactions.

Durability defines the persistence of committed data: once a transaction commits, the data should persist in the database even if the system crashes before the data is written to non-volatile storage.

A **schedule** is a series of (possibly overlapping) transactions.

A **blind write** is when a transaction writes to an object without ever reading the object.

A **dirty read** occurs when a transaction reads a database object that has been modified by another not-yet-committed transaction.

An **unrepeatable read** occurs when a transaction is unable to read the same object value more than once, even though the transaction has not modified the value. Suppose a transaction T2 changes the value of an object A that has been read by a transaction T1 while T1 is still in progress. If T1 tries to read the value of A again, it will get a different result, even though it has not modified A.

A **serializable schedule** over a set S of transactions is a schedule whose effect on any consistent database instance is identical to that of some complete serial schedule over the set of committed transactions in S.

A **recoverable schedule** is one in which a transaction can commit

2. **(Exercise 16.2)** Exercise 16.2 Consider the following actions taken by transaction T 1 on database objects X and Y :

   R(X), W(X), R(Y), W(Y)

   a. Give an example of another transaction T 2 that, if run concurrently to transaction T without some form of concurrency control, could interfere with T 1.

Answer:

a. If the transaction T2 performed W(Y) before T1 performed R(Y), and then T2 aborted, the value read by T1 would be invalid and the abort would be cascaded to T1 (i.e. T1 would also have to abort).

3. **(Exercise 16.3)** Consider a database with objects X and Y and assume that there are two transactions T1 and T 2. Transaction T 1 reads objects X and Y and then writes object X. Transaction T 2 reads objects X and Y and then writes objects X and Y.

   a. Give an example schedule with actions of transactions T1 and T2 on objects X and Y that results in a write-read conflict.

   b. Give an example schedule with actions of transactions T1 and T2 on objects X and Y that results in a read-write conflict.

   c. Give an example schedule with actions of transactions T1 and T2 on objects X and Y that results in a write-write conflict.

Answer:

a. The following schedule results in a write-read conflict:
   T2:R(X), T2:R(Y), T2:W(X), T1:R(X) ...
   T1:R(X) is a dirty read here.

b. The following schedule results in a read-write conflict:
   T2:R(X), T2:R(Y), T1:R(X), T1:R(Y), T1:W(X), T2:R(X) ...
   Now, T2 will get an unrepeatable read on X.

c. The following schedule results in a write-write conflict:
   T2:R(X), T2:R(Y), T1:R(X), T1:R(Y), T1:W(X), T2:W(X) ...
   Now, T2 has overwritten uncommitted data.